



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-412696

XML Format for SESAME and LEOS

J. K. Durrenberger, J. R. Neely, P. A. Sterne

May 1, 2009

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

XML Format for SESAME and LEOS

1	OBJECTIVE AND SCOPE	2
2	REQUIREMENTS.....	2
3	FILE SYSTEM DIRECTORY STRUCTURE.....	3
4	XML DATABASE REPRESENTATION	3
5	EOS XML ELEMENTS	3
5.1	PRIMITIVE ELEMENTS	3
5.1.1	<data> element	4
5.1.2	<dimension> element.....	5
5.1.3	<value> element	5
6	COMPOUND ELEMENTS.....	6
6.1	PRIMARY COMPOUND ELEMENTS	7
6.1.1	<function> element.....	7
6.1.2	<curve> element.....	7
6.1.3	<material> element	8
6.1.4	<directory> element.....	9
6.1.5	<library> element	10
6.1.6	<master_index> element	10
6.2	SECONDARY COMPOUND ELEMENTS	10
6.2.1	<parameter> element.....	11
6.2.2	<grid> element	11
6.2.3	<axis> element	12
6.2.4	<multitable> element.....	12
6.2.5	<individual_phase> element	13
6.2.6	<info> element	13
7	DATABASE ELEMENTS.....	14
7.1.1	<library_id> element.....	14
7.1.2	<reference> element.....	14
7.1.3	<referenced_by> element	15
A.	MATERIAL EXAMPLE.....	17
STANDARD TABLE EXAMPLE.	17
MULTI TABLE EXAMPLE.....	18
B.	FUNCTION EXAMPLE	21
C.	CURVE EXAMPLE	22
D.	LIBRARY EXAMPLE.....	23
E.	MASTER INDEX EXAMPLE	24

Revision History	Author(s)	Comments
1.0	Kevin Durrenberger, Phil Stern, Rob Neely	Version 1 ready for comment by LANL/Sesame team

1 Objective and Scope

The objective of this document is to describe the XML format used by LLNL and LANL to represent the equation-of-state and related material information in the LEOS and SESAME data libraries.

The primary purpose of this document is to describe a specific XML format for representing EOS data that is tailored to the nature of the underlying data and is amenable to conversion to both legacy SESAME and LEOS binary formats.

The secondary purpose is to describe an XML format that lends itself to a “natural” representation in a binary file format of the SESAME, pdb or hdf5 form so that this format and related tools can be used for the rapid and efficient development and implementation of prototype data structures.

This document describes the XML format only. A working knowledge of LEOS and SESAME formats is assumed.

2 Requirements

The following represents high level descriptions of requirements this new XML format is intended to address. Specific requirements may be referenced in justifications in other parts of this document

1. Must be a format that both LANL and LLNL can use within their existing library framework.
2. Complete representation of the relevant physical data in a format consistent with the structure and nature of the data.
3. Consistent formats for similar but distinct data objects.
4. Ability to reference a data element (e.g. grid axis) from multiple locations.
5. Accommodate various grid types.
6. To the extent possible, it must be adaptable for future data layouts not foreseen at the moment.
7. Allow addition of new data – specifically new thermodynamic or related functions that have the same format as existing known functions - without modification to the import/export tools.
8. Support N-dimensional tabular data.

9. Support for multitable/multiphase representations, with boundary curves defining irregular phase boundaries within a grid
10. Structures that are easily validated with an XML schema, or external validation program.

3 *File System Directory Structure*

We assume that the data files comprising the set of EOS data will reside in a Unix like directory structure, although the format and referencing system used does not limit us to this structure. Path separators for systems such as Windows® might need to be addressed if using a coding language which does not handle Unix type path separators (such as C or C++). The location of files and directories has been left up to the individual library maintainer.

4 *XML Database Representation*

The structure of the data itself within the XML EOS data structures can be represented as a hierarchical database structure such as a Unix-like directory structure, but it is not limited to that format. More complicated relational structures can be represented if necessary.

5 *EOS XML Elements*

This XML format recognizes three general classes of element:

1. Primitive element: elements that map directly onto data variables in the data libraries
2. Compound element: elements that correspond to a collection of data variables in the data libraries
3. XML database management element: elements that facilitate management of the XML EOS database

These three general types are described in more detail below.

5.1 *Primitive Elements*

Primitive XML elements consist of logical, byte, integer, longint, float, double or complex scalar variables, or character strings, or arrays of any of these. Primitive XML elements have an obvious and direct mapping to equivalent standard data types in programming languages like C and FORTRAN, and

standard data types in portable file formats such as hdf5 and pdb. More complicated data objects such as structs and common blocks are treated as compound XML elements.

5.1.1 <data> element

Syntax:

```
<data type="data type"
      name="data_name"
      numdims="dimensions"
      innerdims=" inner dim size">

<dimension>
<value>
```

DTD description:

```
<!ELEMENT data (dimension+| value+)>
```

Purpose:

Represents a generic primitive data type. Must contain either zero or more <value> elements or one or more <dimension> elements. The <dimension> elements may be nested to form a multidimensional array. If numdims = 0 or is omitted in <data> then the data is assumed to be a scalar and the <dimension> element is not allowed. The nesting level of the dimension element will be numdims -1 for a multi-dimensional array containing scalar values and numdims for multi-dimensional arrays containing vector return values.

A data element may contain individual data "values" that are themselves vectors or multidimensional arrays. For example, the phase fractions in a multiphase EOS are vectors of dimension numPhases defined on a density-temperature grid. In such cases, the data element must have an innerdims attribute that defines the dimensionality of the data in the inner most dimension. If innerdims=="0" then the array contains scalar value at each final dimension point. If innerdims>0 then the last dimension of the data is a multi-dimensional structure. For example if innerdims=" 1" then the last dimension of data element will be a one dimensional array or vector. If innerdims="2" then the last dimension of the data element will be a 2 dimensional array of values. If the innerdims attribute is omitted it is assumed that the data contains a scalar value and not a multi-dimensional object. If numdims == 0 or is omitted, then innerdims is not allowed.

Attributes:

1. **type:** (required) This is based upon programming language types as follows:
 - byte

- int
- longint
- float
- double
- complex
- dcomplex
- string

Additional types may be added later.

2. **name:** (optional) character string defining the name for the data element. If missing, it defaults to the name of the immediate parent compound element. For example, a `<data>` element will appear inside a `<parameter>` element along with units. The `<data>` element assumes the name of the parameter.
3. **numdims:** (optional) number of array dimensions. Defaults to 0 (scalar) if missing.
4. **innerdims (optional):** This defines the dimensionality of the final dimension of the data element contained by this compound element. If numdims attribute is omitted or equals zero then innerdims is not allowed.

5.1.2 *<dimension> element*

Syntax:

```
<dimension index="0">
  <value>
<dimension>
```

DTD description:

```
<!ELEMENT dimension (dimension+| value+)>
```

Purpose:

For array `<data>` types (those with a `numdims` attribute > 1), the `<dimension>` element groups all of the values for a single dimension of the N-dimensional `<data>` type.

Attributes:

1. **index:** (required) Supplies the 0-based (c-style) index corresponding to the subscript of this dimension of the array data being described in a `<data>` context. At the highest level of nesting, the `index` tag of the `<dimension>` element corresponds to left-most dimension in a multidimensional array $A(i,j,k,l,\dots,z)$. The index at the deepest level of nesting corresponds to the second dimension from the right. The right-most index is set by the `index` of a `<value>` element.

5.1.3 *<value> element*

Syntax:

```
<value index="0"> This is a string </value>
```

DTD description:

```
<!ELEMENT value ( #PCDATA ) >
```

Purpose:

Holds a single scalar value or character string for each `<value></value>` pair. Multiple `<value>` elements will appear in array `<data>` containers.

Attributes:

1. **index:** Is required for numdims greater than zero . Index value for a vector or array corresponding to the 0-based subscript for the rightmost dimension in an N-dimensional array.

6 Compound Elements

Compound XML elements can consist of a collection of primitive XML elements, other compound XML elements and database management elements. Compound XML elements bundle together a related set of information into a single entity. They can be relatively simple, e.g. a parameter named `bulkmod` is a set of two primitive XML data elements inside a `<parameter>` element: one unnamed data element of type `double` containing the value of the bulk modulus, and another named “units” of type `string` that identified the physical units for this value (e.g. “Mbar”, “GPa”, “erg/cc”). They can also be very complicated, for example, the element for the entire data library can contain many compound elements including functions and material information for many individual materials in several distinct libraries.

Compound elements are identified as either primary or secondary. Primary compound elements form complete independent conceptual entities in the EOS data library. They can exist as the sole contents of a file as part of an xml library without requiring an additional xml tag to wrap the data. The data in a primary compound element can stand alone without requiring additional complementary data.

Examples include individual thermodynamic functions, EOS material entries, and complete data libraries.

Secondary compound elements form well-defined components of primary and other secondary compound elements. Although they are “complete” in the sense that they completely define a particular quantity, they are always used in conjunction with other data, and so do not stand alone as independent primary data entities. Examples include information blocks, grid and axis elements, and table data. A table element and a grid element are both required to define a material function, and neither forms a complete entity without the other, hence their designation as secondary compound elements.

Secondary compound elements may not exist as the sole contents of a file in the xml data library; they must always be wrapped in a parent primary data element.

Each compound XML elements has a mandatory `name=""` attribute. Compound XML elements may have additional attributes which can be optional or required.

Compound XML elements can reference other compound XML elements of the same type. See the database management element section for details. When referencing a compound XML element with a different set of attribute values, the values set explicitly in the referencing element take precedence over the values inherited from the referenced element.

6.1 Primary Compound Elements

6.1.1 *<function> element*

Syntax:

```
<function name="name" type="type">
  <reference>
  <referenced_by>
  <parameter>
  <info>
  <grid>
  <data>
```

DTD description:

```
<!ELEMENT function (referenced_by?, (reference| (info?, parameter*, grid, data)) )>
```

Purpose:

Function is the base container for any table. It is designed to be simple and complete while still able to handle relatively complex data setups(i.e. N-Dimensional arrays). It may contain multiple parameter elements but only a single data element.

Attributes:

1. **name:** (required) Contains one of Pressure, Internal Energy, Helmholtz Free Energy, Gibbs Free Energy, Entropy, etc. (same as what's specified in the material index). However the naming convention is flexible and should be left open.
2. **type:** (optional) Contains one of Total, Nuclear, Electron, Ion, Cold Curve, etc. (same as what's specified in the material index). This is optional in the sense that if the name is sufficiently unique this may be omitted.

6.1.2 *<curve> element*

Syntax:

```
<curve name="curve name" type="curve type">
  <reference>
```

```
<referenced_by>
<info>
<parameter>
<data>
```

DTD description:

```
<!ELEMENT curve (referenced_by? , (reference | ( info?, parameter*, data)) )>
```

Purpose:

To provide a unique structure for storing curves which do not fit the definition of a function. It does not have an overlaying grid that a function would have. It would allow for multiple parameter elements but only a single data element.

Attributes:

1. **name:** The name of this curve(i.e. phasecurve)

6.1.3 <*material*> element

Syntax:

```
<material name="name" version="version">
  <reference>
  <referenced_by>
  <info>
  <library_id>
  <parameter>
  <multitable>
  <function>
```

DTD description:

```
<!ELEMENT material (referenced_by? , (reference | ( info?, library_id, parameter*, (multitable |
  function+)) )>
```

Purpose:

The complex material object contains all the information to access the various material properties and all function tables related to a specific material. We have 2 different material structures a multi-table or the standard table representation.

Attributes:

1. **name:** (required) This will be the actual name of the material and not the library defined numeric symbol. For example, if this is hydrogen in an LEOS table we would not use 10 or 11 to define the name, but rather “hydrogen”.

2. **version:** (required) This is used to further define the material based on additional information from the corresponding library.

6.1.4 <directory> element

Syntax:

```
<directory name="directory name">
  <info>
  <parameter>
  <directory>
```

DTD description:

```
<!ELEMENT directory ( info? , parameter*, directory* )>
```

Purpose:

The <directory> tag serves two purposes. First, within the XML representation, it provides a way to create general hierarchical data structures that will convert predictably to a corresponding hierarchical data representation in our target binary formats. Second, it provides a general format that can be used to represent source data in a recognized binary file format (e.g. hdf5 or pdb) that does not conform to our recognized set of primary and secondary compound data elements.

It is generally preferable to define new primary and secondary compound elements when necessary to represent any new data structures, together with prescriptions for converting this new data structure into our binary data file formats. The <directory> tag should therefore not be used in final, mature versions of EOS data libraries. It is nevertheless useful, especially during development and prototyping phases, to be able to specify directly how the data should be represented in the binary file format. The <directory> tag facilitates this type of rapid development.

The name of the directory and the location in which it will be created are defined by the “name” attribute. The location is assumed to be relative to the current position in the file unless the name starts with a “/”, in which case it is interpreted as an absolute location. If parent directories do not exist, they are created. The current position is determined by the parent tag of the <directory> tag provided that resolves to a well-defined location. If the parent tag is a material, the directory is created in the material directory; if it is a function, the directory is created in the function directory. The directory may already exist, in which case it will remain in place and may have contents added to it by nested <dimension> and <parameter> tags.

Attributes:

1. **name:** The name assigned to this directory-like object (directory, group, folder etc.) in any non-XML representation into which we are going to convert the data.

6.1.5 *<library> element*

Syntax:

```
<library name="library name">
  <reference>
  <referenced_by>
  <material>
```

DTD description:

```
<!ELEMENT material (referenced_by? , (reference | material+ )>
```

Purpose:

High level container for each library

Attributes:

1. **name:** (required) unique library name such as SESAME or LEOS

6.1.6 *<master_index> element*

Syntax:

```
<master_index  name="database">
  <library>
```

DTD description:

```
<!ELEMENT master_index (library+ )>
```

Purpose:

Highest level container to group various libraries together

Attributes:

1. **name:** (optional) Not sure what a name on this would be but it could possibly need one.

6.2 Secondary Compound Elements

6.2.1 <parameter> element

Syntax:

```
<parameter name="">
  <data>
```

DTD description:

```
<!ELEMENT parameter (data+)>
```

Purpose:

This is a wrapper for any piece of data. It can be contained by the following elements: < properties>, <table>, <function>, <multitable>, <curve> and <axis>. It supplies the name to a specific piece of data and allows for additional data elements which should be associated with this specific parameter. For example a data value for a parameter and a definition of the units for this parameter value would be assigned this manner:

```
<parameter name="rhomin">
  <data name="units" type="string">
    <value>g/cc</value>
  </data>
  <data type="double">
    <value>some double value</value>
  </data>
</parameter>
```

Attributes:

- 1. name:** (required) Name of the parameter. There is really no restriction, however we should be more descriptive than what is currently contained in the libraries.

6.2.2 <grid> element

Syntax:

```
<grid name="nameof grid" type="grid type">
  <reference>
  <referenced_by>
  <axis>
```

DTD description:

```
<!ELEMENT grid (referenced_by?, (reference | axis+) )>
```

Purpose:

Container which describes the encompassing axes of a table. It can be contained only by an <function> element.

Attributes:

1. **name:** (optional) Name assigned to this grid
2. **type:** (required) Contains one of cartesian, spherical, unstructured, etc.

6.2.3 <axis> element

Syntax:

```
<axis name="axis name">
  <reference>
  <referenced_by>
  <data>
```

DTD description:

```
<!ELEMENT axis (referenced_by? , (reference | data )>
```

Purpose:

Container for an individual axis in a table. It can be contained only by a <grid> element.

Attributes:

1. **name:** (optional) Name assigned to this axis

6.2.4 <multipitable> element

Syntax:

```
<multipitable name="multipitable name">
  <info>
  <parameter>
  <material>
  <individual_phase>
```

DTD description:

```
<!ELEMENT multipitable ( info?, parameter*, material, individual_phase+ )>
```

Purpose:

This is the design for the LEOS multi-table library. The multi-table considers each individual phase as a separate material and a single material which has the flattened table information thus the <material> element under the <multipitable> element. This can only be contained by a <material> element.

Attributes:

1. **name:** (optional) Any name the library wishes to assign the <multipitable> element.

6.2.5 <individual_phase> element

Syntax:

```
<individual_phase name="phase name">
  <info>
  <curve>
  <material>
  <parameter>
```

DTD description:

```
<!ELEMENT individual_phase ( info?,curve*, parameter*, material )>
```

Purpose:

This provides access to the individual phases directly associated with a specific multi table setup. It uses the <curve> elements to access such things as the phase curves and outer curves and defines the phase itself as a <material> element .

Attributes:

1. **name:** (required) this is the name of the individual phase (Examples would be liquid, bcc, ...)

6.2.6 <info> element

Syntax:

```
<info name="name">
  <date_created>
  <date_last_modified>
  <author>
  <citations>
    <citation>
  <comments>
    <comment>
```

DTD description:

```
<!ELEMENT info (date_created?, date_last_modified?, author?, citation?, comments?)>
```

Purpose:

Info blocks are used to record one or more comments concerning the object. Info blocks can be inserted inside any compound XML element but are not required. This allows for the separation of informational data which, while important to understanding the origins of the base data, is not data with respect to the equation of state.

Attributes:

1. ***name***: (optional) This might be used to keep the association with a specific compound element.

Sub element information

The sub elements of info are defined by name and not generalized as parameters. They all will consist of PCDATA to allow for free input of strings values within the elements. Additional elements can be added if needed later.

7 Database Elements

7.1.1 <*library_id*> element

Syntax:

```
<library_id>
  <library_entry library="src library name" map="map to name">
```

DTD description:

```
<!ELEMENT library_id (library_entry+)>
```

Purpose:

Provide additional information about containing library's structure. It contains one or more `<library_entry>` elements and has no attributes. `<library_entry>` will contain 2 attributes which will allow the code to map this material back to a specific library.

Examples:

For a material hydrogen which maps back to LEOS L10:

```
<library_id>
  <library_entry library="leos" map="L10"/>
</library>
```

For a material hydrogen which would map back to SESAME 5250:

```
<library_id>
  <library_entry library="sesame" map="5250"/>
</library>
```

7.1.2 <*reference*> element

Syntax:

```
<reference path="file path" type="reference origin type"  
query="Xpath query"/>
```

DTD description:

```
<!ELEMENT referenced EMPTY>
```

Purpose:

It acts effectively as an “include” in a complex element, allowing multiple data types to reference the same data. This can save space by getting rid of duplicate data, as well as ensuring consistency. This is contained by either a `<referenced_by>` element or most complex elements. The query needs to be of a valid IEEE XPath query syntax. Although not all parsers currently support XPath query language there are sufficient open source tools available and if need be a parser can be written to perform the simple queries which we will be using.

Attributes:

1. **path:** (optional) This is the path to the file containing the referenced XML. If omitted, it defaults to the current file.
2. **type:** (optional) This is either relative, absolute or library_base. In the case of library_base we take the path from the base of the containing library. This would be the default if this is empty or omitted. If path is omitted this will be ignored.
3. **query:** (optional) This is a standard XPath query to perform on the given document. If this is omitted then it is assumed the entire document is be referenced in the path.

Examples:

```
<reference path="L10/total_pressure.xml" type="library_base"/>
```

This says to open the file L10/total_pressure.xml using the base of this library as path starting point. We will use the entire document within. This would be the most common use of complex object referencing an external complex object.

```
<reference path="L10/total_pressure.xml"  
query="//grid/axis[@label='temperature']"/>
```

This is an example of an axis which references a common axis in some function file. This allows us to point to a specific axis within the file. This also makes the assumption since the type is omitted that the path starts at the library base.

7.1.3 `<referenced_by>` element

Syntax:

```
<referenced_by>  
<reference>
```

DTD description:

```
<!ELEMENT referenced_by (reference*)>
```

Purpose:

Provide a mechanism by which references to an object can be updated if a file is moved or altered in a manner which makes the referring reference query or path invalid.

Attributes:

None

Material Example

Standard table example.

```
<?xml version="1.0" encoding="UTF-8"?>

<material name="vanadium Flattened multitable" version="9230">
  <info>
    <comments>
      <comment>
        Vanadium flattened from liq and solid tables based
        phonon DFT, LMTO cold
      </comment>
    </comments>
  </info>
  <parameter name="bulkmod" >
    <data name="units" type="string">
      <value>string value</value>
    </data>
    <data type="double">
      <value>1.745500000000000e+12</value>
    </data>
  </parameter>
  <parameter name="comp_a" >
    <data type="double">
      <value>5.0941499999999978e+01</value>
    </data>
  </parameter>
  <parameter name="comp_frac" >
    <data type="double">
      <value>1.000000000000000e+00</value>
    </data>
  </parameter>
  <parameter name="comp_z" >
    <data type="double">
      <value>2.300000000000000e+01</value>
    </data>
  </parameter>
  <parameter name="ecoh" >
    <data name="units" type="string">
      <value></value>
    </data>
    <data type="double">
      <value>2.310000000000000e+11</value>
    </data>
  </parameter>
  <parameter name="rho0" >
    <data name="units" type="string">
      <value></value>
    </data>
    <data type="double">
      <value>6.209399999999959e+00</value>
    </data>
  </parameter>
```

```

</parameter>
<parameter name="t0" >
    <data name="units" type="string">
        <value></value>
    </data>
    <data type="double">
        <value>3.024900000000009e+02</value>
    </data>
</parameter>
<parameter name="num_comp" >
    <data type="int">
        <value>1</value>
    </data>
</parameter>
<parameter name="formula" >
    <data type="string">
        <value>D</value>
    </data>
</parameter>
<function name="pressure" type="total">
    <reference path="L9230/Pt.xml"/>
</function>
<function name="entropy" type="total">
    <reference path="L9230/Et.xml"/>
</function>
<function name="pressure" type="cold">
    <reference path="L9230/Pc.xml"/>
</function>
<function name="energy" type="cold">
    <reference path="L9230/Ec.xml"/>
</function>
<function name="St">
    <reference path="L9230/St.xml"/>
</function>
<function name="temperature" type="melt">
    <reference path="L9230/Tm.xml"/>
</function>
<function name="Cs">
    <reference path="L9230/Cs.xml"/>
</function>
</material>

```

Multi table example

NOTE: Omitted material parameters

```

<?xml version="1.0" encoding="UTF-8"?>

<material name="vanadium multitable" version="9230">
    <info/>

    <multitable >
        <parameter name="rhomax">
            <data name="units" type="string">
                <value>string value</value>
            </data>

```

```

<data type="double">
    <value>2.11474722487499989e+01</value>
</data>
</parameter>
<parameter name="rhomin">
    <data name="units" type="string">
        <value>string value</value>
    </data>
    <data type="double">
        <value>5.63932593300000029e+00</value>
    </data>
</parameter>
<parameter name="tempmax">
    <data name="units" type="string">
        <value>string value</value>
    </data>
    <data type="double">
        <value>2.000000000000000e+04</value>
    </data>
</parameter>
<parameter name="tempmin">
    <data name="units" type="string">
        <value>string value</value>
    </data>
    <data type="double">
        <value>1.000000000000000e+00</value>
    </data>
</parameter>

<individual_phase name="bcc">
    <info>
        <date_created>date</date_created>
        <date_last_modified>date</date_last_modified>
        <author>author or source name</author>
        <citations>
            <citation>character string</citation>
        </citations>
        <comments>
            <comment>character string</comment>
        </comments>
    </info>
    <curve name="PhaseCurve">
        <reference path="M9230/IP1/phasecurve.xml"/>
    </curve>

    <material name="vanadium" version="I9232">
        <!--
        make this name and version take precedence over retrieved
        reference name and version reference to material IXXXX
        -->
        <reference path="I9232/material.xml"/>
    </material>

    </individual_phase>
    <individual_phase name="liquid">
        <info>

```

```
</info>

<curve name="PhaseCurve">
    <reference path="M9230/IP2/phasecurve.xml"/>
</curve>

<material name="vanadium"  version="I9233">
    <reference path="I9233/{material}.xml"/>
</material>
</individual_phase>
<material name="vanadium flattened" >
    <reference path="L9230/material.xml">
</material>
</multitable>

</material>
```

A. Function Example

```
<?xml version="1.0" encoding="UTF-8"?>

<function name="pressure" type="total" innerdims="0">
    <referenced_by>
        <reference path="L9230/material.xml"
            query="/material/state/function[@name='pressure' and
                @type='total']"/>
    </referenced_by>
    <info></info>
    <grid type="cartesian">
        <axis label="temperature">
            <reference path="L9232/{function}.xml"
                query="//grid/axis[@label='temperature']"/>
        </axis>
        <axis label="density">
            <reference path="L9232/{function}.xml"
                query="//grid/axis[@label='density']"/>
        </axis>
    </grid>
    <data numdims="2" type="double" innerdims="0">
        <dimension index="0">
            <value index="0">double string value</value>
            <value index="1">double string value</value>
            <value index="2">double string value</value>
        </dimension>
        <dimension index="1">
            <value index="0">double string value</value>
            <value index="1">double string value</value>
            <value index="2">double string value</value>
        </dimension>
    </data>
</function>
```

B. Curve Example

```
<?xml version="1.0" encoding="UTF-8"?>

<curve name="PhaseCurve" >
  <referenced_by>
    <reference path="L9230/material.xml"
      query="/material/multitable/curve[@name='phasecurve' and
        @type='parametric']"/>
  </referenced_by>
  <info></info>
  <parameter name="units">
    <data numdims="1" type="string" innerdims="1" >
      <dimension index="0">
        <value index="0">units string</value>
        <value index="1">units string</value>
      </dimension>
    </data>
  </parameter>
  <parameter name="axisLabels">
    <data numdims="1" type="string" innerdims="1" >
      <value index ="0">label string</value>
      <value index ="1">label string</value>
    </data>
  </parameter>
  <data name="data"  numdims="1" type="double" innerdims="1" >
    <dimension index="0">
      <value index="0">double string value</value>
      <value index="1">double string value</value>
    <dimension>
      <dimension index="1">
        <value index="0">double string value</value>
        <value index="1">double string value</value>
      <dimension>
        <dimension index="1">
          <value index="0">double string value</value>
          <value index="1">double string value</value>
        <dimension>
      </data>
    </curve>
```

C. Library Example

Example library file

```
<?xml version="1.0" encoding="UTF-8"?>

<library name="leos">
    <material name="vanadium flattened multitable" version="9230">
        <reference path="L9230/material.xml"/>
    </material>
    <material name="vanadium multitable" version="109230">
        <reference path="M9230/material.xml"/>
    </material>
</library>
```

D. Master Index Example

Master index file example

```
<?xml version="1.0" encoding="UTF-8"?>

<master_index name="primary">
    <library name="LEOS" >
        <reference path="LEOS/library"/>
        <material name="SESAME">
            <reference path="SESAME/library.xml"/>
        </material>
    </library>
</master_index>
```